Coding in a Distributed Team: Testing, Reviewing, Sharing and Merging Code Without Going Crazy

Andrew Bennetts

1 Introduction

In my work for Canonical Ltd, I'm part of a team of nearly twenty Launchpad[1] developers spread around the globe. I also contribute to the Bazaar distributed version control tool[2] and the Twisted networking framework[3]. In each case, major contributors work in a wide variety of timezones.

Common problems with distributed development include:

- making sure that developers on one side of the world aren't blocked for a day on colleagues on the other side of the world
- allowing all developers to work on a functional shared codebase
- and checking that code works before checking it into the shared codebase

I will describe some of the technical tools Launchpad, Bazaar and Twisted developers use to solve these problems: the distributed version control system Bazaar and the Patch Queue Manager which controls code merges. I will also discuss some of the QA procedures used by the team, including automated test suites, and mandatory code reviews and how these are integrated with the version control system.

2 The Trunk

There is a main development branch all developers on base their work on — the "trunk".

Tests must always pass on the trunk

The golden rule is that **tests must always pass on the trunk**.

If tests don't pass, then developers are impeded. At best they're annoyed as failures irrelevant to their changes clutter their test runs, and at worst it stops productive work entirely while the software doesn't run and they try to figure out why. In a distributed team, the developer that caused the problem probably won't be around for hours. The simplest policy is that if a change on the trunk broke the test suite, it will be reverted.

If you don't have an automated test suite, then you should! Automated test suites give you some assurance that your code has a basic level quality that you can check just by hitting a button. Every feature and bugfix that is tested are features and bugfixes you can be reasonably confident are still working while you make other changes. A developer never needs to wait for someone in another timezone to wake up and explain why something isn't working (or how to tell if it's working!) and how to fix it, so long as that something has tests and the tests are getting run.

They also alert you to portability problems: when something in your environment changes (say you've upgraded Python, or your database, or maybe your whole OS), you can run the test suite to immediately find out if you've broken something. This is particularly helpful when developers all have subtly different configurations on their development machines, as is the case when they're working from home rather than in a centrally managed office environment.

Because tests are required to pass on the trunk, when a developer breaks an interface, they cannot land the

1https://launchpad.net/
2http://bazaar-vcs.org
3http://twistedmatrix.com/

change until they fix all uses of that interface in the codebase first (or provide backwards compatibility).

Enforcing tests pass on trunk with PQM

Developers shouldn't have to run the test suite every time they update a checkout of the trunk. It'd be an annoying waste of time. But they should still be confident that all tests are passing on the trunk, so there needs to be some way to enforce, or at least make it easy to check, the health of the trunk.

A common approach is the use of a tinderbox or buildbot that developers can check — if the latest build web page is green, the trunk is safe to use. Twisted uses this method.

Launchpad and Bazaar uses the Patch Queue Manager[4] (PQM) tool instead. No developer is able to commit directly to the trunk, instead they have to tell the PQM process to merge branches to the trunk on their behalf.

The PQM is configured to run the test suite *before* a committing a change to the trunk, and only accept the commit if it passes. This is similar to a "pre-commit" hook in SVN, but PQM processes requests asynchrously, so clients don't need to stay connected while the entire test suite runs.

Here's an example of submitting a merge request to PQM, using the pqm-submit plugin for Bazaar[5]:

```
$ vim foo.py
$ bzr commit -m "Twiddle bits in foo.py"
$ vim bar.py
$ bzr commit -m "Twiddle bits in bar.py too." $ bzr push
$ bzr pqm-submit -m "Twiddle foo.py and bar.py"
```

The `pqm-submit` command sends a GPG-signed email to the PQM instructing it to merge the changes in your branch into the trunk. The email will look something like:

```
Subject: Twiddle foo.py and bar.py

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
echo star-merge url://to/trunk url://to/my/branch
...
```

PQM will use GPG to check that the sender of the email is allowed to commit to the specified branch, merge the submitted branch into a checkout of the trunk, and then run a pre-commit hook, e.g. `make check`. If the hook succeeds, the merge is committed. When the commit is complete, or when it is rejected, PQM will send an email with the results: either a simple "merge successful" message, or a log of the failed merge (e.g. a report of merge conflicts, or failed tests). If the commit succeeded, it also mails a commits mailing list, so other developers are notified of changes. A developer can "fire and forget", and start work on their next task while PQM runs the test suite.

A PQM and a buildbot can be complementary — PQM might just run tests on one or two core platforms, and then a buildbot can do test runs on a wider variety of platforms and configurations. It's a balance between "absolutely must work" and "should work, but we're prepared to be a little bit tolerant."

Coping with inter-dependent projects

Launchpad has dependencies on various other projects. Some of these are internal projects, and others are external ones like Zope, SQLObject and Twisted. We use config-manager[6] to build Launchpad checkouts with these dependencies checked out in a "sourcecode" subdirectory:

(root of Launchpad checkout)
  README
  ...

4http://bazaar-vcs.org/PatchQueueManager
5https://launchpad.net/products/bzr-pqm
6http://people.ubuntu.com/~robertc/config-manager/

```
sourcecode/
        ...
        sqlobject/
        twisted/
        zope/
```

(Although most developers just use rsync to copy a pre-built up-to-date checkout off our development server.)

Every now and then we patch or upgrade our copy of these dependencies. This obviously carries a significant risk of causing Launchpad's tests to stop passing — and because PQM requires that tests pass before anyone can commit to Launchpad's trunk, this would halt all changes to trunk.

To avoid this problem, we use PQM to manage commits to the dependencies. The pre-commit hook we use for the dependencies requires not only that the tests for the branch involved passes, but that Launchpad's tests also pass. This way we can only upgrade a dependency if it doesn't break the Launchpad.

Actually, because some dependencies rely on other dependencies, we keep it simple: for most branches, we require PQM to run "make check_merge", a rule that runs make check for Launchpad, and each of the dependencies managed in the "sourcecode" directory.

3 Mandatory Code Reviews

The Launchpad, Bazaar and Twisted projects have all independently decided that all changes must be reviewed before they can "land" on the trunk.

Mandatory code reviews are another way of ensuring the code on the trunk has a certain basic level of quality. Not just that the code is as defect-free as possible, but also that it remains as easy for other developers to work with as possible. Code reviews catch various problems: bugs, coding style inconsistencies, unreadable code, code with insufficient tests.

They also share knowledge throughout a team, particularly about infrastructure in the existing code that other developers might not be aware of: "You could use the FooHelper class here instead of writing that yourself." They also help a project notice when there's a need for a piece of infrastructure that doesn't exist yet — if a reviewer sees different pieces of code struggling with the same problems, perhaps it's time to share that work, or think of a more general solution. A distributed development team doesn't gather around a water cooler very often, so you need some other way to share knowledge of the problems and solutions people are finding. Code reviews help notice problems and solutions people haven't thought to email about.

Reviews also encourage smaller branches and smaller landings, because it's much easier and faster to get a 300 line diff reviewed (and passing review!) than a 3000 line diff.

Launchpad manages pending reviews with on a wiki page. Developers add an entry with the URL to their branch, and a summary of the changes. These are allocated daily to a reviewer in the review team (a subset of the developers), who generally reviews it within 2 days. Discussions about reviewed code take place on a dedicated mailing list. Commits to the trunk must have "[r=reviewer]" in the commit message. PQM has been configured to reject commits without a reviewer in the commit message.

Bazaar uses its main developer discussion list as the review forum. Developers post messages to the list with "[MERGE]" in the subject, and any other developer is welcome to review and give their opinion. Approval from two other developers is required for most changes, indicated by a "+1" in a reply.

Twisted uses its Trac ticket tracker for reviews. Each branch is expected to have a corresponding ticket, whether for a bug or a feature. Developers add the "review" keyword to the ticket to add it to the review queue. Other developers periodically check the review queue for unassigned reviews, and tickets needing reviews are also periodically announced by a bot on the IRC channel. Commits to trunk must say "Closes #nnnn" in them. The Trac instance will automatically close the tickets based on these commit messages.

In all cases, there's a forum for discussion of reviews (whether mailing lists or a ticket), and all developers are able to participate in the discussions.

All three projects expect changes to make corresponding changes to the test suite.

4 Bazaar: Distributed Version Control

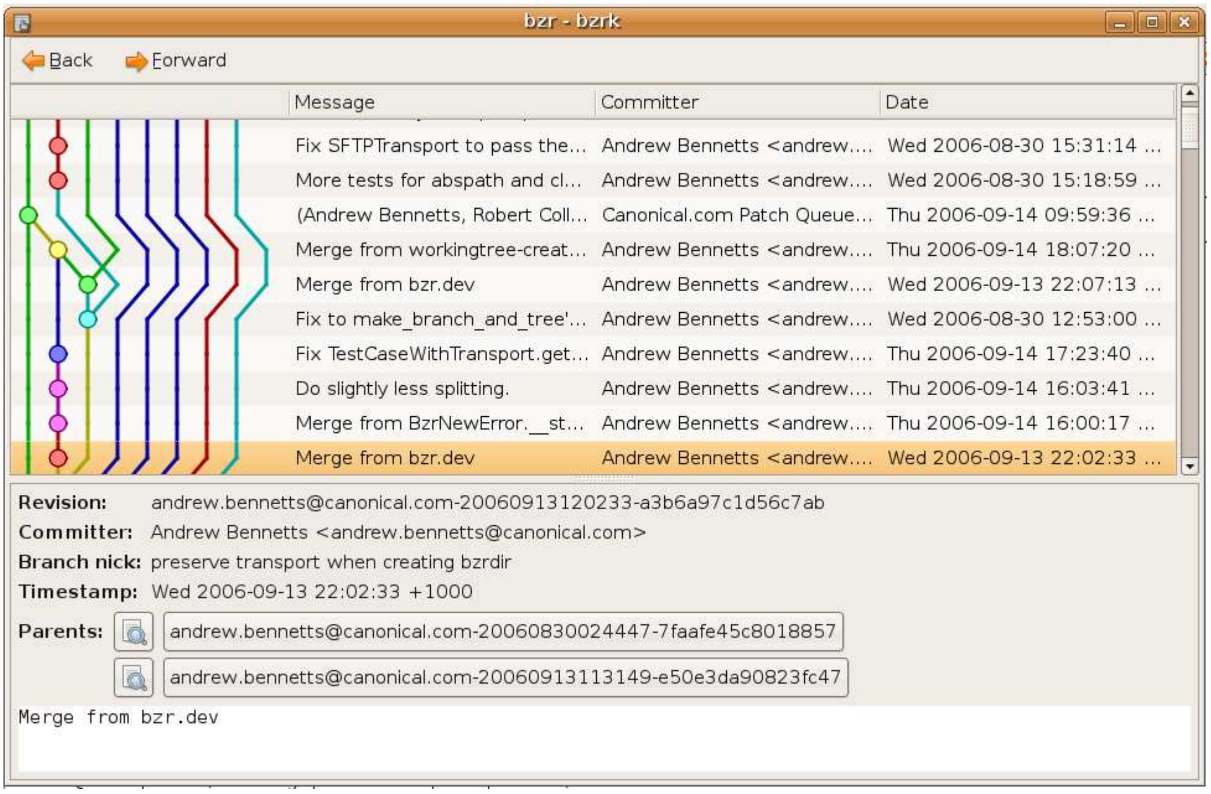
Launchpad and Bazaar use Bazaar as their revision control system. Bazaar is a *distributed* revision control tool, developers can branch, commit, diff, log, annotate, and merge while disconnected. This is extremely valuable when travelling, but is also a generally pleasant model to work with.

Distributed version control tools like Bazaar typically have excellent merging support, because it's fundamental to being distributed that every checkout is effectively a new branch. This is particularly helpful if you like to have a checkout per feature or bug you're working on — each checkout is inherently a complete branch that you can manage with the full suite of revision control commands, not just a limited "commit everything or nothing."

A revision control tool that makes creating and merging branches as simple as possible is important. Often you have a branch not yet merged to the trunk (e.g. because it's pending review) and you want to start a new branch with the work in your unmerged branch. If your revision control tool makes dealing with branches of branches awkward, difficult or tedious, then it's making your job is harder than it should be. A similar situation arises when two or more developers are working on a feature together (although perhaps in different timezones) — they will want to be able to merge back-and-forth frequently and easily.

Good branching and merge support also encourages developers to make branches with single, specific purposes, e.g. "Add feature X" or "Fix bug #1234". This in turn makes it easier to review changes. It's possible to do this with e.g. Subversion too, but it requires more discipline and more effort by developers.

A related effect of distributed revision control tools is that they encourage much smaller commits within a branch, too. This is great when you are digging through the history of some code with "annotate" trying to figure out why it is the way it is, because there is a finer-grained history for you to look through.



Message	Committer	Date
Fix SFTPTransport to pass the...	Andrew Bennetts <andrew....	Wed 2006-08-30 15:31:14 ...
More tests for abspath and cl...	Andrew Bennetts <andrew....	Wed 2006-08-30 15:18:59 ...
(Andrew Bennetts, Robert Coll...	Canonical.com Patch Queue...	Thu 2006-09-14 09:59:36 ...
Merge from workingtree-creat...	Andrew Bennetts <andrew....	Thu 2006-09-14 18:07:20 ...
Merge from bzr.dev	Andrew Bennetts <andrew....	Wed 2006-09-13 22:07:13 ...
Fix to make_branch_and_tree'...	Andrew Bennetts <andrew....	Wed 2006-08-30 12:53:00 ...
Fix TestCaseWithTransport.get...	Andrew Bennetts <andrew....	Thu 2006-09-14 17:23:40 ...
Do slightly less splitting.	Andrew Bennetts <andrew....	Thu 2006-09-14 16:03:41 ...
Merge from BzrNewError.__st...	Andrew Bennetts <andrew....	Thu 2006-09-14 16:00:17 ...
Merge from bzr.dev	Andrew Bennetts <andrew....	Wed 2006-09-13 22:02:33 ...

Revision: andrew.bennetts@canonical.com-20060913120233-a3b6a97c1d56c7ab
Committer: Andrew Bennetts <andrew.bennetts@canonical.com>
Branch nick: preserve transport when creating bzrdir
Timestamp: Wed 2006-09-13 22:02:33 +1000
Parents: andrew.bennetts@canonical.com-20060830024447-7faafe45c8018857
andrew.bennetts@canonical.com-20060913113149-e50e3da90823fc47

```
Merge from bzr.dev
```

Illustration 1: Screenshot of "bzr visualise" of Bazaar, showing branching and merging

Some advantages of Bazaar specifically:

- it's written in Python,

- it’s easy to use,
- the “uncommit” command,
- the “shelve” command (part of the bzrtools[7] plugin),
- PQM already works with it.

5 Conclusion

Based on my experiences with Launchpad, Bazaar and Twisted, I recommend projects with distributed developer teams:

- Require the trunk to *always* build and pass tests. Enforce it with a tool like PQM if you can, otherwise use a buildbot so the state of the trunk is always immediately available to developers.
- Require code reviews to commit to the trunk. These not just improve quality of the code, they share knowledge about the problems and solutions the developers are encountering.
- Use a distributed revision control tool. They cope will with the extensive branching and merging distributed development tends to cause.

7http://bazaar-vcs.org/BzrTools